ulm university universität ulm

Ulm University | 89069 Ulm | Germany

Faculty of Engineering,Computer Science and Psychology
Institute of Databases and Information Systems

Design, concept and implementation for an electronic and mobile patient health record of tinnitus affected patients using the iOS platform

Master's Thesis at Ulm University

Submitted by:
Carmen Vazinkhoo
carmen.vazinkhoo@uni-ulm.de

Reviewer:
Prof. Dr. Manfred Reichert
Dr. Winfried Schlee

Supervisor:
Dipl. Inf. Rüdiger Pryss

2015

Version September 16, 2015

Satz: PDF-LaTeX 2$_\varepsilon$

Abstract

Tinnitus is increasingly affecting individuals who report that they hear a whistilng or other noise without any external sound source. Currently, there is no treatment that completely eliminates tinnitus. However, there are ways to allay this, for example by questionnaires or therapies. On the basis of auditory tests changes can be documented in the perception of tinnitus. Through observation of event triggers and amplifiers, appropriate actions can be taken. For all of these approaches, the *TinnitusNavigator* app has been developed which supports the patient in dealing with tinnitus and brings the patient closer to a relief of symptoms. Users will find an easy navigation since the iOS-ware Guidelines were considered in the design. Using the Core Data framework of Apple supported the realisation of the model from the Model View Controller pattern. At the end the outcome of this work is a working application.

Note of thanks

At this point I would like to thank everyone who has supported me during the preparation of this master's thesis and during my student days.

With special thanks to Prof. Dr. Manfred Reichert who has always supported me during my academic studies.

To my advisor Rüdiger Pryss, who supported me through his feedback and knowledge during the master's thesis.

My greatest thanks dedicates to my parents and family which not only funded my studies and enabled me the possibility to study, but also constantly showed a great interest in my work as well as supported me wherever they could.

Contents

1

Introduction

Affected person describe the tinnitus as a noise, whistling, hissing or rattling in the ear and in the head. There is no external sound source and no other person can hear the tinnitus of the patient. The annoying noise or whistling appears particular unpleasant when there is surrounding silence. Sometimes it occurs suddenly and fades away immediately. But in some cases it persists and can torture the patient. Ear noises or whistles are symptoms which can be compared to pain or fever. Tinnitus can again lead to numerous concomitant symptoms like sleep problems, poor concentration, anxiety and depression. Also, tinnitus is a warning signal of pressure both mentally and physically. It is assumed that inflammation of the ear or of the respiration track, noise damage, organic sickness like autoimmune diseases as well as viral infections are possible causes but the actual creation mechanism is still unsolved. Therefore it is often difficult to cure the root cause. Discussed influences and risk factors are alcohol, nicotine, various medicines,

food and as already mentioned, stress. However, the symptoms can be eradicated with help of combined treatment methods, self-help groups or hearing aids.

1.1 Motivation

There is a set of methods for treating tinnitus. For example, one method uses different questionnaires each with a different focus. It takes time until the filled out questionnaires of the patients are evaluated and so the feedback is not immediately available. Finding the right treatment and likewise keeping track of the results takes a lot of effort. Patients have to rely on the documentation of the doctor and wait for the results. If the patient want to check up their health record, they have to make an appointment and wait to get a copy. The tinnitus is always abstract to the affected person and complex to understand relation between treatments and results.

Mobile applications for the medical and psychological context are complex to implement [1, 2, 3]. To improve this situation, in 2008 the Tinnitus Research Initiative was founded to develop effective treatments for various forms of tinnitus in order to provide a relief for the patients. From that the TrackYourTinnitus project has been formed which enables the affected person to monitor individual variations of the tinnitus perceptions with their own smartphone. This provides a systematic documentation over several weeks.

Suitable for this task are smartphones as they are omnipresent and can be used everywhere at any time. Patients can record an occurring event right away and also look up certain data. As there is a set of different therapies, a smartphone can support keeping track of the therapies.

An instrument to capture differences in the hearing perceptions are audiometries. With that, hearing thresholds and changes can be documented as well as be compared. In this way everything is stored in one place and can be fetched when needed, such as at a doctor's appointment to discuss progress of a therapy or to find a suitable new one.

Therefore the goal of this thesis is to implement an app, called TinnitusNavigator, which provides input options for events, audiometric measurements and therapies. Also the

app includes different questionnaires answered and saved. The user can access the data any time and anywhere it is required. TinnitusNavigator app can been seen as a mobile health record of the patient to navigate through the tinnitus handling.

1.2 Structure of this thesis

This section provides an overview of the structure of this work. Chapter 2 lists all necessary medical fundamentals and summarizes previous implementations within the TrackYourTinnitus projects. Based on this knowledge, the requirements analysis, which is separated into functional and non-functional requirements, is elaborated in chapter 3. The TinnitusNavigator app is introduced with its functionality and its structure in detail within chapter 4. Afterwards follows in chapter 5 a general overview of the application's architecture with its components and also the data model and the relation of the app to the corresponding server. Chapter 6 gives further insights into the implementation as well as implementation aspects. Then, the defined requirements are matched with the level of development in chapter 7. Chapter 8 provides an outlook on further improvements and chapter 9 concludes this thesis.

2

Fundamentals

This chapter presents the fundamentals that are required for a better understanding of tinnitus. It deals with its definition, present types, explains questionnaires and defines audiometry. Finally, related implementations for the TrackYourTinnitus projects are introduced.

2.1 Definition of tinnitus

Tinnitus is defined as a phantom perception of sound if an external sound source is missing [4]. *Tinnitus aurium* is the Latin word for *ringing in the ears* (*tinnire* = sound, ringing) [5]. As a medical term tinnitus means a perception of acoustic nerve impulses that are generated at any point of the auditory pathway and do not necessarily come from external acoustic stimuli. At the same time tinnitus is a symptom of disturbed

auditory perception, almost always as a consequence or associated symptom of hearing impairment. It is distinguished from many more rare sound varieties existing in the body such as vascular or muscular clacking noises. This may be perceived also from unaffected individuals objective tinnitus, even if they actually belong to the body's intrinsic noises. Tinnitus affects about 10% of all adults in the world [6]. Thus it is starting to be recognized as a global health problem.

2.2 Kind of tinnitus

The *tonal appearance* of tinnitus far from uniform, because a tinnitus can occur in a variety of forms [7]. It describes pure tones at different frequencies, tonal mixtures and narrowband or broadband noise. In general, however, most tinnitus forms are high-frequency whistling, which can be a result of hearing loss in the high frequencies. It may vary in its intensity, at least in the subjective perception and finally can be constant or pulsating. However, for therapeutic considerations, it has been proven that one can not only rely on such subjective descriptions of tinnitus, and it is more important to follow a certain classification system which distinguishes the following forms of tinnitus.

Objective/subjective tinnitus

Sometimes ringing in the ears can be heard from the outside, by doctors under investigation, which are very rare (0.01%). Actually, however, the patient hears pathologically increased body self noise. This is called *objective tinnitus*. The normal hearing person perceives nerve impulses that occur in the body, usually near the ear. Many autonomic body functions with noises occur, such as breathing, heart and bowel function. Generally, these are not perceived as these are experienced as naturally and are usually very quiet. Only if they become louder like the heartbeat during exercise, they will be consciously heard. If such noises occur regularly in the body but are not immediately classified by the body as such noises, then this can be perceived as *objective tinnitus*.

The vast majority of tinnitus forms are *subjectively* in nature, for example, they can not be heard from outside. The body's own sound sources are not detectable with current methods of audiometric examination. However, the corresponding tinnitus are not imaginary. To date researchers have not yet succeeded in finding appropriate methods to uniquely measure the electrophysiological effects of tinnitus. This is because the actual noises in the ear are never more than 5-10dB above the hearing threshold. Therefore, stimulations of its intensity would only be audible directly to the auditory nerve, but it would be too invasive to directly attempt to measure the response of the auditory nerve.

Tinnitus with/ without hearing loss

One of the main distinguishing characteristics is the coincidence of tinnitus and accompanying hearing loss. Rarely is the hearing in tinnitus patients completely normal. Rather, tinnitus arises precisely when the hearing gets diminished. Its frequency is almost always the frequency of the largest hearing loss. This applies in particular to the high-frequency noises in the ear, which are the associated with symptoms of weakness in the high frequencies or deafness. If the hearing is completely normal, the ear noise is then considered as a result of a general over stimulation and an incorrect processing in the auditory pathway.

Acute/ chronic tinnitus

Duration and persistence of ear noise are of great importance, especially with regard to therapeutic possibilities. An acute tinnitus (first occurring) is one that often disappears spontaneously or after appropriate therapies. If the tinnitus symptoms persist longer than 3 months, it is called a *chronic tinnitus*. However, this depends on the meaning that the ear noise has for each patient.

Compensated/decompensated tinnitus

Choices for therapy and need for treatment depends on how each patient can deal with their tinnitus and how much they suffer. According to current surveys, 25% of all Germans have already experienced tinnitus at least once and 13% hear the tinnitus above a longer period. However, only 2% are significantly affected by the noises. This means that the majority of people who hear a ringing in the ears, feel that it is not very disturbing even if it is present permanently. In this case, tinnitus is compensated by normal habituation processes so regardless of temporal phases or the duration of its presence, the ear noise rarely disturbs the patient. In this case, tinnitus is compensated by normal habituation processes so regardless of temporal phases or the duration of its presence, the ear noise rarely disturbs the patient. If tinnitus is accompanied by a network of listening issues triggering emotional responses, a normal habituation is prevented. This gives rise to a *suffering on tinnitus* situation where tinnitus threatens to decompensate or is already decompensated, the symptoms dominate the patient, and their quality of life may be greatly restricted. This possible development is carried out regardless of whether tinnitus occurs in the inner ear, in auditory nerve, or is generated in the brain stem or central auditory processing.

2.3 Questionnaires and severity levels

Both experimental and clinical measurements of tinnitus are central concerns of modern tinnitus research and therapy [8]. Until today, however, there is no reliable measurement method available for localization and quantification of tinnitus. Therefore subjective methods, such as psychoacoustic comparative measurements and the self-assessment of patients are the basis of deliverable data for the detection of the tinnitus severity [9, 10]. These tools are especially helpful in capturing the multidimensional nature of tinnitus distress and to detect the biopsychosocial aspect of tinnitus through a wide range of issues [11, 12]. These include questions about tinnitus distraction, concentration, sleep problems, hearing problems, worries about the future, catastrophizing, psychosomatic stress factors, etc., which can be answered by the patients in varying degrees [13, 14]. A

number of differentiated tinnitus questionnaires to capture the severity levels have been developed [15]. Below are the tinnitus questionnaires that are used in the app:

Mini Tinnitus Questionnaire

The purpose of this questionnaire is to find out whether the noises in the ears/head have had any effect on the patient's moods, habits or attitudes [16]. Some questionnaires are problematic because they require a relatively large number of questions which are needed to determine a global distress measure. If the available time for investigation is restricted and other measurements are also need to be completed, this could be a disadvantage. Therefore, there is a growing need for a quicker and compact measure of overall tinnitus distress. The primary goal is to obtain excellent psychometric qualities.

All twelve questions of the *Mini Tinnitus Questionnaire* were selected under strictly defined psychometric criteria. Questions were only considered if they were greatly correlated with the general score. The questionnaire indicates changes of symptomatology. This is important because analysing therapy results using the same questionnaire is an important component of the therapy process. Most central and characteristic aspects of tinnitus distress are represented by the twelve questions. The *Mini Tinnitus Questionnaire* is more useful for chronic tinnitus, since psychological distress in acute patients may be temporary and of lower prognostic value.

Tinnitus Sample Case History Questionnaire

Questionnaires can request information about the history and descriptive characteristics of the patient's tinnitus or tinnitus related conditions [17]. An international body agreed that a question list for *Case History Questionnaires should* be generated which should contain those questions common to many of the questionnaires (and structured interviews) in latest use in order to reach comparability The *Tinnitus Sample Case History Questionnaire* contains 35 questions which can be used in its entirely or modified. This app will use the *Tinnitus Sample Case History Questionnaire* as it is.

Worst symptom

There are a number of symptoms reported by many patients (for example insomnia, concentration difficulties, anxiety) which can become worse with tinnitus [18]. Of this, however, almost every patient has one symptom that is worst for him. This questionnaire aims to identify this symptom.

Create a profile

This questionnaire obtains the most important information about the current situation of the patient [19]. These questions help to determine how strong the patient suffers from his ear noise and what measures have already been taken.

2.4 Audiometry

For the diagnostic evaluation and subsequent therapeutic recommendation a thorough objective audiometric testing is always desired in order to recognize and differentiate diseases, especially of the middle or inner ear, as well as the auditory nerve and central auditory processing [5]. However, the lack of objective detection methods leads the practitioners to try to measure the tinnitus by using pitch and intensity comparisons.

The sound threshold should be determined with special care, especially as the affected patients may indicate different levels od hearing losses or - more rarely - a normal or even remarkably normal hearing threshold. Much effort has to be made while trying to detect the type of tinnitus: sound or noise, in what frequency and at what volume. Generally, tinnitus patients are a challenge to even experienced audiometry assistant regarding the accuracy and reproducibility of measurements.

If the hearing threshold is determined the patient will be offered a comparison sound, which is beginning with the frequency of the largest hearing loss, about 10dB above the threshold, to accurately determine the tinnitus frequency, This will be changed in pitch until it becomes about the same as the sound in the ear.

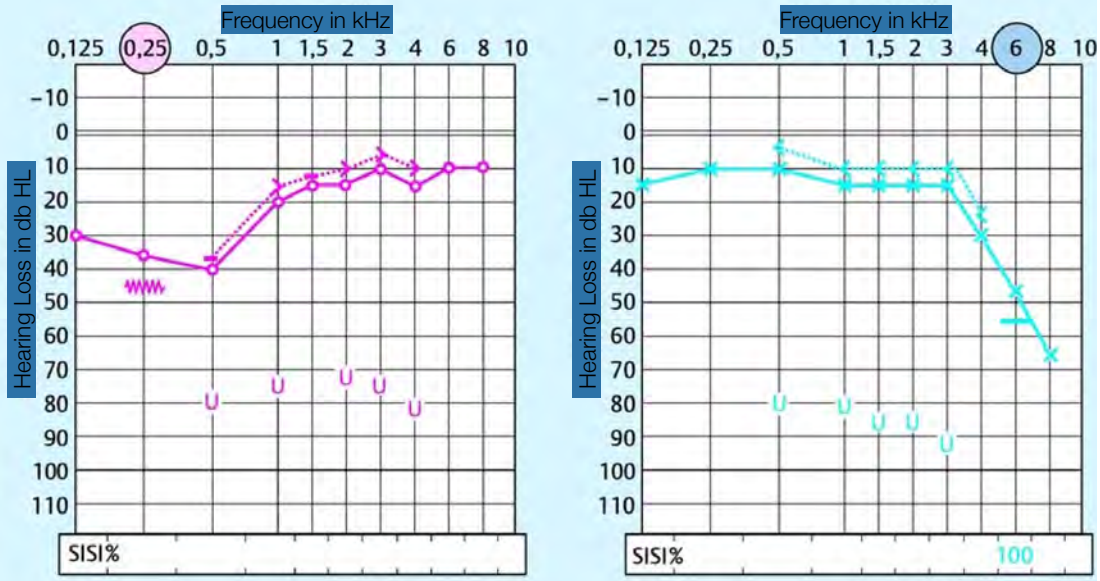

Figure 2.1: Example for a documentation of a tinnitus and the discomfort threshold [5]

The actual loudness of ear noise is generally about 5-10dB above the hearing threshold, even if the patient's tinnitus appears subjectively louder.

The relation of tinnitus loudness to hearing threshold must be carefully documented. Amazingly, the measured level is often unchanged even if, for example after a treatment the subjective loudness appears decreased. That's an indication of the adaptive abilities of central tinnitus processing.

The values found for pitch, timbre and loudness of tinnitus are entered in the audiogram of each ear (see figure 2.1). The pitch affiliation is identified in the frequency coordinate with a circle. A tonal tinnitus is entered as a small zigzag. The discomfort threshold (marked as U) is often noticeablely low.

2.5 Related work

The TinnitusNavigator app belongs to the TrackYourTinnitus research project of the Tinnitus Research Initative and of the Institute of Databases and Information Systems at

the Ulm University [20]. In this context, the following products are developed: Servers for each type of app, Website, TrackYourTinnitus app (for iOS & Android) and TinnitusNavigator app (only for Android) [21].

The TrackYourTinnitus app supports the patient to understand the tinnitus by asking specified question in order to register variation of perception, which can only be done in an app not on the website [18]. Main function is a questionnaire which monitors the perception that reminds the user in irregular periods to fill out a short questionnaire [22].

An extension of the TrackYourTinnitus app is the TinnitusNavigator app which can be used to compared to the patient's health records [19]. The app contains the questionnaires, input options for results of an audiometry measurement, events and therapies [23]. The server of the Android app is still under development [24]. Both apps have their own server which synchronizes the users data.

3

Requirements Analysis

In this chapter the requirements of the mobile application, the TinnitusNavigator, are described. These requirements are divided into functional and non-functional requirements and summarized in table 3.1.

3.1 Functional requirements

This section shows the functional requirements of the app. The main features of the app are shown, which should be provided to the user. The following section explains the functional requirements in detail.

FR 1: **Sign up:**
The user can not make use of the app without a user account, because the user's

data and the questionnaires are saved on the server. A user account is necessary to get access to the server. It should be possible to register without leaving the app.

FR 2: **Log in:**
If a user has an account, it should be possible to sign in to the sever through the app. Afterwards, the data of the user, which are stored on the server, can be downloaded into the app and can be displayed.

FR 3: **Log out:**
It should be possible to log out. Either another user wants to log into the app or the user himself wants to end its connection to the server.

FR 4: **Reset password:**
Did a user forget the password or just can't remember it there should be a way to reset it from within the app instead of opening the browser separately.

FR 5: **Operate without internet connection:**
There are situations that the user may not have an access to the internet, like on journeys or inside of buildings. If the user is already logged in and has pulled the initial data like the questionnaires, the app should still work independently of the internet connection. The user should be informed that there is currently no internet connection but also be able to use the app like before.

FR 6: **Synchronize local data with server data:**
If the app loses the connection to the server (like in offline mode) the modified data should be stored on the phone and automatically be synchronized as soon as the connection to the server is re-established.

FR 7: **Display all kinds of questionnaires from the server in app:**
The questionnaires from the server have different types of answer options. The app should be able to display the answer options dynamically. This way new or changed questionnaires on the server will always be displayed correctly in the app.

FR 8: **Fill out questionnaires:**
The user should be able to answer the questionnaires in the app and see which questionnaires are already filled out and are synchronized with the server.

FR 9: **Enter/show/delete events:**
It is important to keep track of the events of a patient in relation to their tinnitus. Therefore it should be possible to enter such an event, like a tinnitus attack or an appointment, view this event and delete it if necessary. Additionally, a calendar view that includes all events would increase the usability.

FR 10: **Enter/show/delete therapies:**
Also it is important to maintain records of the kinds of therapies a patient is currently in or has completed. That's why it should be possible to enter a new therapy / treatment, view this therapy and delete it if necessary.

FR 11: **Enter/show/delete measurements of audiometry:**
The perception of the tinnitus can vary over time. Also, the results of therapies can only be measured with audiometry. To keep track of the changes of the hearing perceptions,it should be possible to enter a new measurement, for both ears, view such measurements and delete them if necessary. Sometimes the standard frequencies are not enough, therefore the user should be able to enter custom ones. For a better visualisation the result should be displayed in a diagram for each ear.

FR 12: **Update data automatically and manually:**
The data on the phone should synchronize automatically with the server, but the user should also have an option to do the synchronization manually.

3.2 Non-functional requirements

Next, requirements are introduced that are primarily important for appearance and operation of the app as well as for data protection. Like the functional requirements, the non-functional requirements are listed below with a detailed description.

NFR 1: **No saving of email addresses or ip addresses:**
Due to legal restrictions it is not allowed to store the email address of a user without a permission. Also it is not lawful to store the ip address of the user while accessing the app. This should be considered during the implementation.

NFR 2: **Use colour of TrackYourTinnitus app:**
The main colours of the TrackYourTinnitus app are blue and green. These colours should also be used as it associates that these apps are belonging together.

NFR 3: **Following iOS design guidelines:**
As the TinnitusNavigator will be an iOS app, the iOS design guidelines should be followed. This makes it also easier for the user to use the app without being confronted with a whole different and unfamiliar design.

NFR 4: **Intuitive user interface:**
The user should be able to navigate quickly to the desired destination. Also the user should be presensted with similar user elements and interfaces despite of its logic in the background. Tasks should be completed fast.

NFR 5: **Make app scalable as much as possible:**
As the data on the server grows or is modified, the app should be able to handle it. For example, when an additional questionnaire is added to the server, it should be displayed in the app without making any changes.

NFR 6: **Release in Apple App Store:**
The app would be useless if nobody could use it, hence it should be released in the Apple App Store after the implementation is completed.

No.	Description	Kind of requirement
FR 1	Sign up	functional
FR 2	Log in	functional
FR 3	Log out	functional
FR 4	Reset password	functional
FR 5	Operate without internet connection	functional
FR 6	Synchronize local data with server data	functional
FR 7	Display all kinds of questionnaires from server in app	functional
FR 8	Fill out questionnaires	functional
FR 9	Enter/show/delete events	functional
FR 10	Enter/show/delete therapies	functional
FR 11	Enter/show/delete measurements of audiometry	functional
FR 12	Update data automatically and manually	functional
NFR 1	No saving of email addresses or ip addresses	non-functional
NFR 2	Use colour of Track Your Tinnitus project	non-functional
NFR 3	Following iOS design guidelines	non-functional
NFR 4	Intuitive user interface	non-functional
NFR 5	Make app scalable as much as possible	non-functional
NFR 6	Release in Apple App Store	non-functional

Table 3.1: Summarizing table with all requirements

4

Introducing the app

This chapter introduces the TinnitusNavigator app from a user's perspective. The first section 4.1 describes the individual functions based on the app flow illustrated in figure 4.1. Section 4.2 is about the server with which the app is communicating.

4.1 General Structure

The TinnitusNavigator app opens with a login view, since the app can not be used without a user account. If the user is logged in, a tab menu is shown with the following five tabs: *Events*, *Questionnaires*, *Therapies*, *Audio* and *More*. In the *Events*, *Therapies* and *Audio* tabs, the user can add new entries, view details or delete them. The *Questionnaires* tab shows all available questionnaires from the server which the user can fill out or has already filled out. As usual the imprint, about and licenses elaborations can be found

in the *More* tab. Also included in the *More* tab are the logout option and a link to the TrackYourTinnitus app in the Apple App Store.

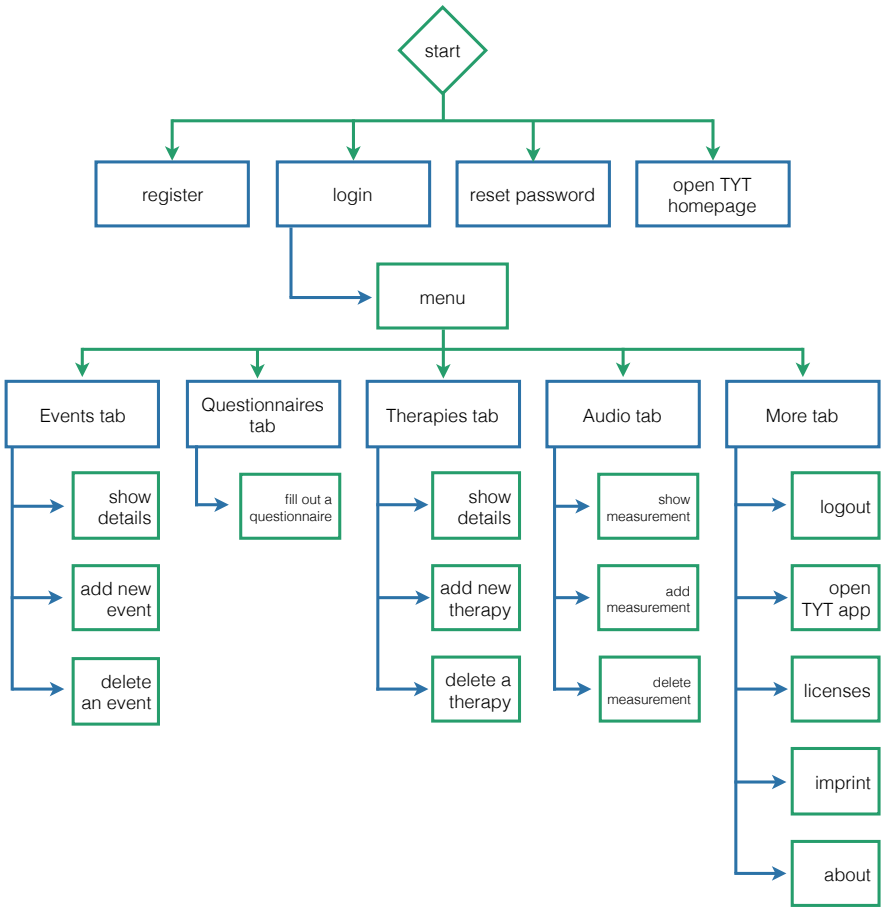

Figure 4.1: The app flow of the TinnitusNavigator app

4.1.1 Login screen

On the first start of the app the login screen is shown. If the user has an account but has forgotten the password, then there is also a reset button which, like the registration button, opens a mobile view inside the app. The register view is the mobile view of the project homepage which is included in the app. That way the user does not need to leave the app for the registration.

Does the user have an account but forgot the password, then there is also a reset button which, like for the registration, opens a mobile view inside the app. The user can stay in the app again.

For login the user has to type in its user name and password. If they are not correct or one of the two is missing, an error message is shown and the user can try again. On success, the tabbed menu is shown.

For more information about the project that this app belongs to, there is a button that leads to its website. This website is also like the other displaying websites integrated in the app.

All mentioned views can be seen in the figure 4.2.

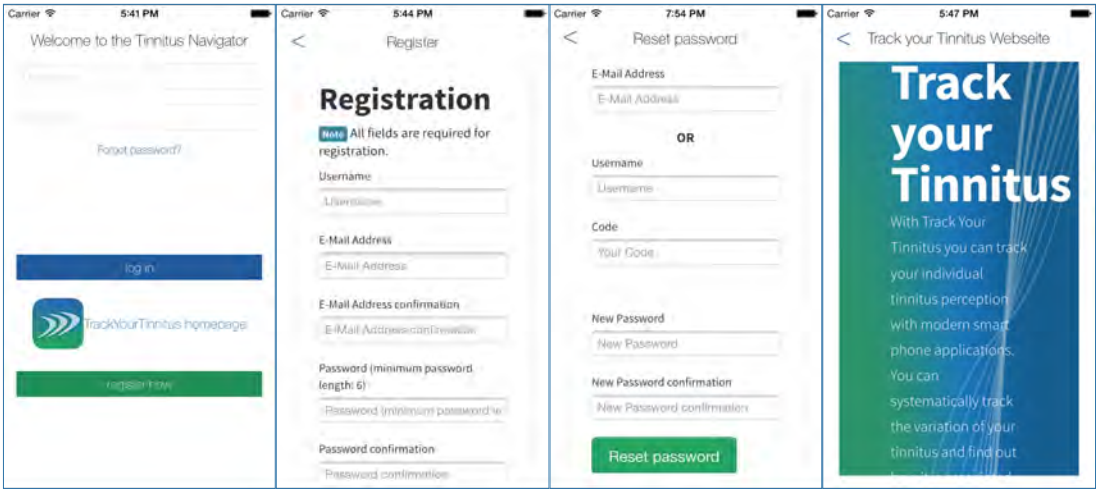

Figure 4.2: Welcome view, register view, reset password view and TrackYourTinnitus website view

4.1.2 Events tab

An event is an occurrence that is related to the patient's tinnitus. It can be an appointment, mental illness, tinnitus attack or something else. As mentioned in chapter 2, it is important to keep track of a variety of information surrounding the patient to find out

which influences increases the perception of the tinnitus. Figure 4.3 shows all associating views.

To add a new event its name, date and type have to be filled in. The type can be selected from a spinning wheel which has the above mentioned options. A description can be entered optional. If one of the mandatory entries is missing an error message is shown.

The user can get an overview of all previously inserted events in the *Events* tab. They are indicated with their name and date. As that overview is about events, the sorting of them is by date.

If an event is selected from the overview, details of this are shown. The details are in the same order as they are filled out to make it easier to orientate and to find the desired information faster.

An event can be deleted in the overview section. For that the event has to be shifted to the left. On the right side appears a delete button. If the delete button is pressed, the event is deleted. To not delete the event, then the event can be pushed back to the right and the delete button disappears again.

4.1.3 Questionnaires tab

As shown in section 2.3 questionnaires make up one of the main tools of the treatments to determine the impact of the tinnitus. Questionnaires that are used by the project are: create profile, *Mini Tinnitus Questionnaire*, *Tinnitus Sample Case History Questionnaire* and worst symptom. All questionnaires are accessed from server and displayed dynamically depending on the answer types of each question which can be seen in figure 4.4.

To fill out a questionnaire, it is selected from the overview. The questions are always coloured white on a green background. Each question has, as mentioned, a different answer type: radio buttons, text, check buttons, scale values or date of birth.

If a questionnaires is fully filled out, it can not be filled out again. This is indicated by the green check mark to the right of the questionnaire's name in the overview. Otherwise there is an grey arrow on the right.

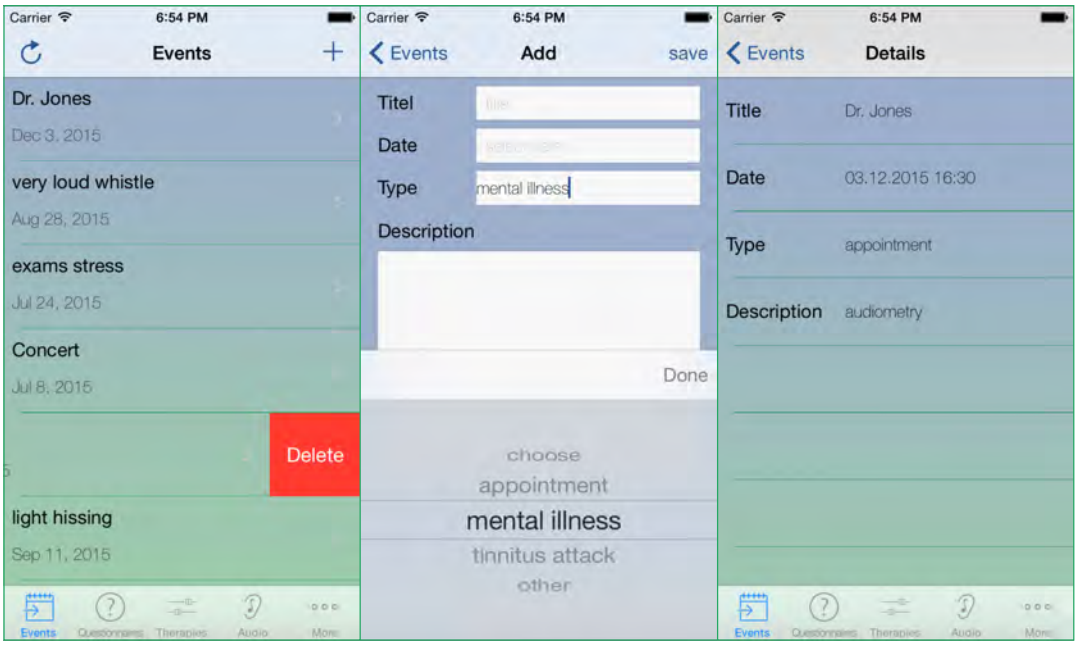

Figure 4.3: All associating views if *Events* tab

4.1.4 Therapies tab

A therapy can help the patient reduce the perception of tinnitus. It is important to maintain an overview of therapies which are administered. The *Therapies* tab is structured as in figure 4.5. The following types are currently available in the app:

- Auditory stimulation
- Biofeedback
- Dental treatment / treatment of the jaw
- Hearing aid
- Medical treatment
- Neurofeedback
- Physical therapy
- Psychotherapy

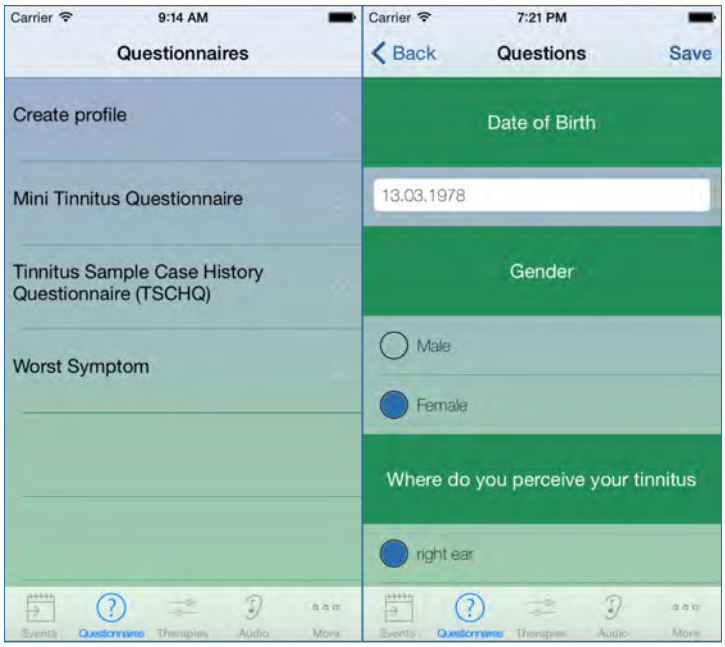

Figure 4.4: *Questionnaires* tab with its overview and the beginning of the *Tinnitus Sample Case History Questionnaire*

- Sport/physical exercise
- Tinnitus masker / tinnitus noiser
- Tinnitus Retraining Therapy (TRT)
- Traditional Chinese Medicine
- Transcranial Magnetic Stimulation (TMS)
- Other

To add a therapy, the user has to type in a name, start date, end date, type of therapy and a note. Name, start date and type are mandatory whereas end date and note are optional. The therapy type can be selected using a spinning wheel.

All inserted therapies are listed in the overview. Each therapy is indicated by its name and type. This time the focus is on the therapy, that is why the overview list is sorted by types.

With selecting a therapy from the overview its details are shown. The details view is set up similar to the add view, again for better orientation and faster finding of information.

The delete procedure is the same as that presented in the *Events* tab. To delete, swipe the therapy it to the left and push the delete button. Swipe the therapy back to the right to abort the deletion.

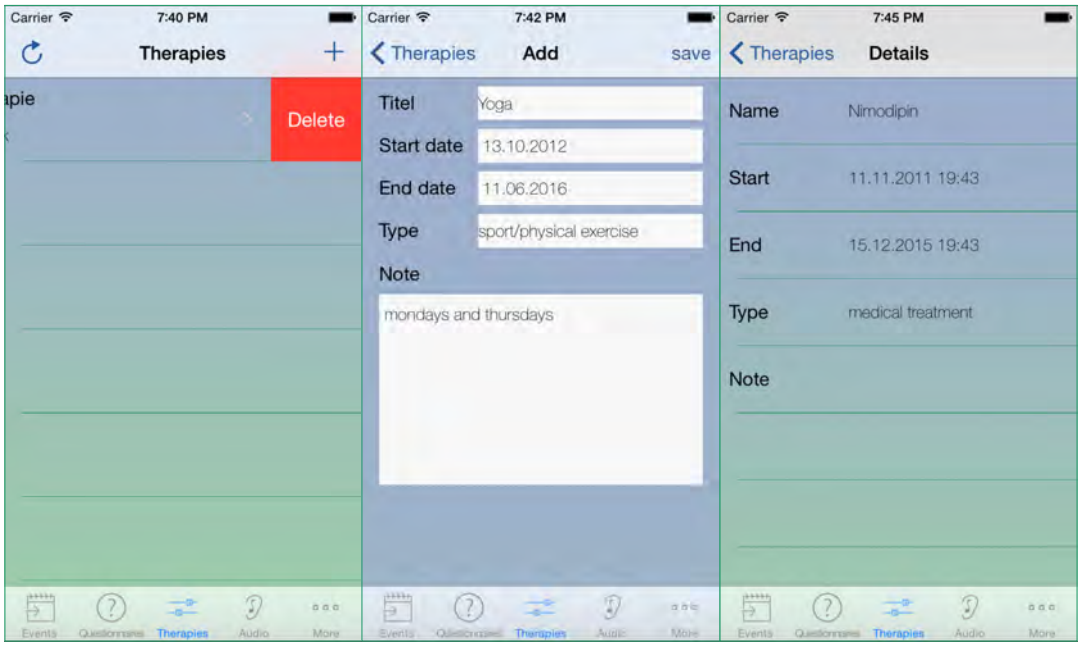

Figure 4.5: *Therapies* tab and its associating views

4.1.5 Audiometry tab

Audiometry a help determine changing in the hearing. It is simple for the patient to keep the results of the audiometry together with the events, therapies and questionnaires.

Adding a measurement requires a date and all measured results of the left and right ears for all frequencies. Standard frequencies are 125Hz, 250Hz, 500Hz, 1000Hz, 2000Hz, 4000Hz and 8000Hz.

All measurements of the audiometry are indicated in the overview by its date. As there is only this indicator, the overview is sorted by it.

As in the previous tabs, the details for each measurement of the audiometry can be accessed by selecting one from the overview. The details view is similar to the adding view.

Figure 4.6 displayed all above mentioned views.

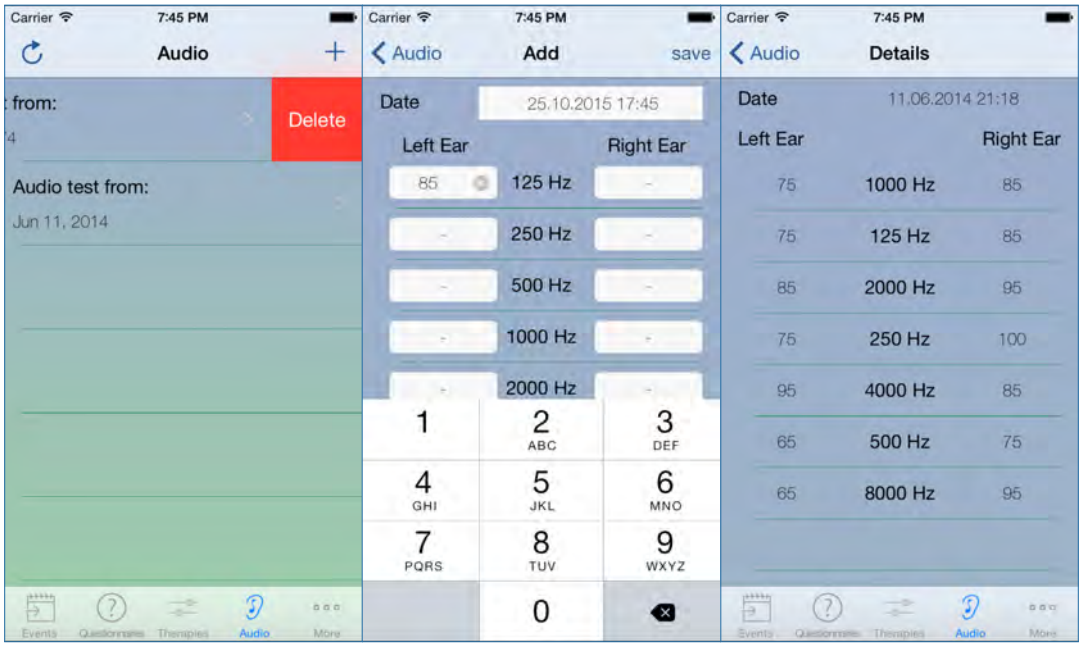

Figure 4.6: *Audio* tab with overview, adding view and details view

4.1.6 More tab

The *More* tab is the last tab in the app. It contains a table with five elements that are shown in 4.7. First element is logout. By selecting it, the user is requested, if logout should really perform. Confirming this question logs the user out. This additional question prevents to log out the user by accident.
The second element is used to open the belonging TrackYourTinnitus app in the Apple App Store. With this other app the patient can track the change of its tinnitus. Also it considers daily routines and activities.
Next element is the licenses, which contains the licenses of the used frameworks in the

app which are described more in detail in section 6.3.

Fourth element is the imprint which contains the legally prescribed declaration of origin.

Last element is about the TrackYourTinnitus project and its contributors.

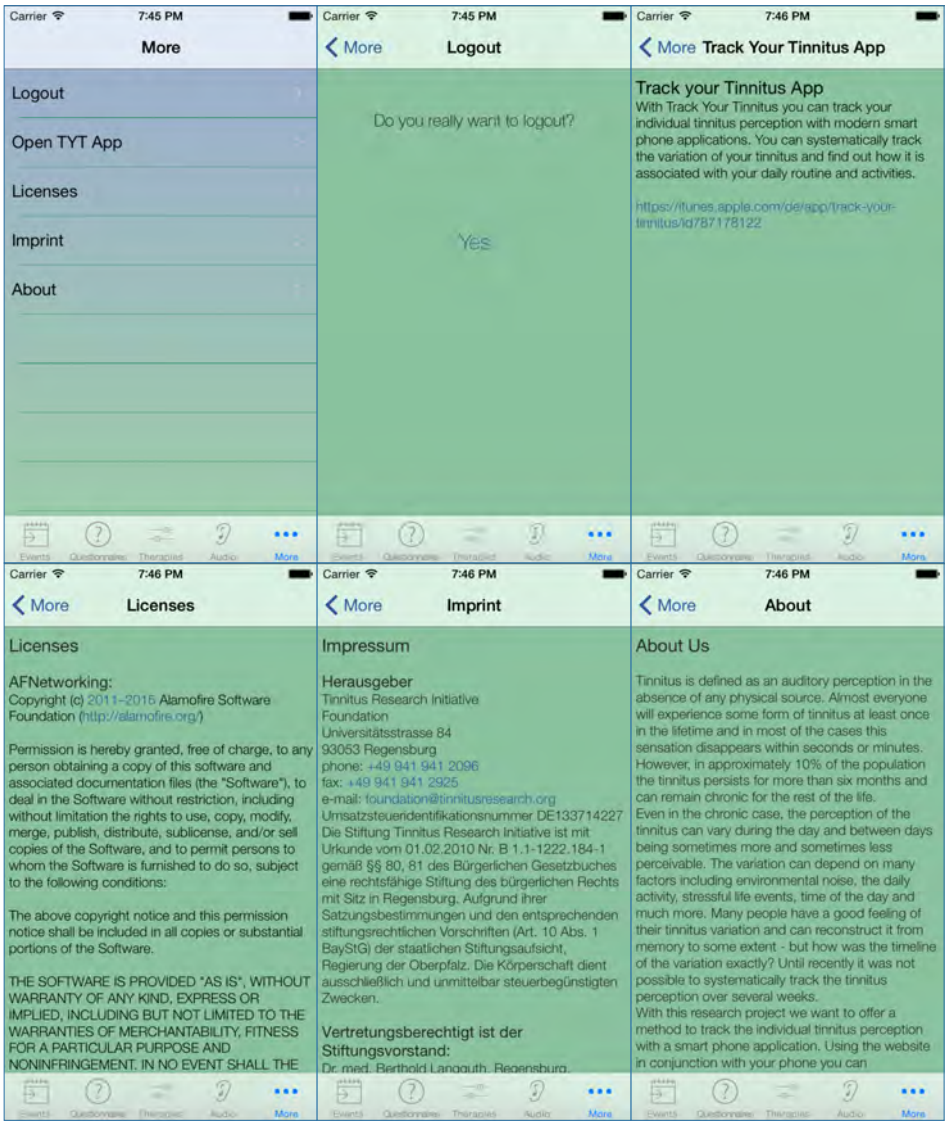

Figure 4.7: *More* tab with views of all five elements

4.2 Server

App is communicating with a server which is the same like of the Android TinnitusNavigator app. The user's data, which are entered through the app or on the website, are stored in the server. In case the user changes his/her phone the data are not lost as they are synchronized with the server. After every change made in the app a date exchange with the server is performed. Additionally, a refresh button can be pushed to force the exchange. If the server is unreachable a message informs the user and the modified data are flagged for the exchange when the server is reachable again.

5

Architecture

This chapter is about the architecture of the TinnitusNavigator app. A brief overview section is followed by a detailed description of the components of the architecture. At the end, the relation between the app and the server is presented.

5.1 General structure

The TinnitusNavigator app supports all iPhones that are running with iOS 7 and later. As measured by the Apple Store on August 17th 2015, 86% of devices are using iOS 8 and 13% are using iOS 7 [25]. Less than 2% are using older versions. That is why the app supports versions starting from iOS 7. All classes have the prefix *TNA* which stands for *TinnitusNavigator app* and guarantees a unique class name within the app [26]. The

prefix has to be three letters or more because the two-letter options are reserved for use by Apple; for example,like *UI*, which stands for *User Interface*.

5.1.1 View Controller & Table View Controller

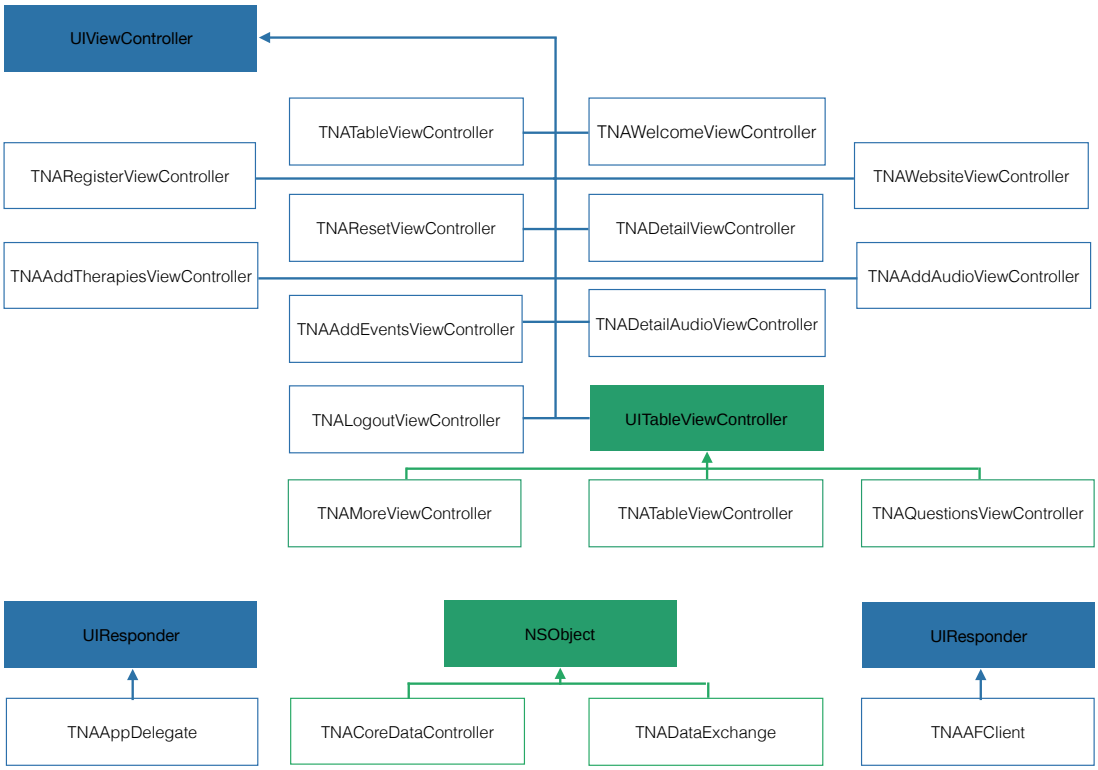

Figure 5.1: View controllers and table view controllers

Every view controller inherited from *UIViewController* excepts that a table can be found there which it inherits from *UITableViewController*. In figure 5.1 is the architecture of the view controller and other classes. At the start of the app the *TNATableViewController* is loaded which checks if the user is logged in. If not the *TNAWelcomeViewController* handles the login. It loads the view, manages the whole login mechanism and saves the received access token of a successful log in. More details about the log in process is described in section 6.3 as the *AFOAuth2Manager* framework is used to realize this process. From this view the *TNAWebsiteViewController* can be called to get to

the website of the TrackYourTinnitus project which is loaded in a *UIWebView* inside of the app. Also the *TNARegisterViewController* which shows the register web view and the *TNAResetViewController* which shows the web view to reset the password can be invoked from the *TNAWelcomeViewController*. If the user successfully logged in then the *TNATableViewController* is shown. It manages whether the user is logged in and the dynamic table views of the tabs: *Events*, *Questionnaires*, *Therapies* and *Audio*. These four do not have their own user interface as the same interface is simply reused. They all have the same structure. To configure properties for each view, they need a unique *User Defined Runtime Attribute* which allows it to address the right view [27]. All views have in common that they are showing *NSManagedObjects* (cf. 5.1.3). That is why the entity name of this objects is selected as the attribute. Depending on the attribute the corresponding data is fetched, refreshed, added, deleted or displayed. For switching to another view segues are defined which specify which view is loaded. *TNAAddEventsViewController* can be called to add a new event, *TNAAddTherapiesViewController* to add a new therapy and *TNAAddAudioViewController* to add a new audiometric measurements. By selecting a row of the table view in the *TNATableViewController* the details for this object are displayed. Event and therapy objects have the same structure for the details hence they are both handled in the *TNADetailViewController*. Audiometric measurements are composed of several objects so *TNADetailAudioViewController* needs special handling. All corresponding objects for the details view have to be fetched and sorted. The *TNAQuestionsTableViewController* needs to fetch the questions for the selected questionnaire and implement the right answer types. Last tab is connected to the *TNAMoreTableViewController*. It contains a static table which is defined in the storyboard and needs less effort for realizing the view. All called views from the *TNAMoreTableViewController* are modal views which are managed by the storyboard because they show static text and have no user interaction. Only the *TNALogoutViewController* is called to show the log out view and to execute the log out process.

Additionally to the view controllers are these classes: *TNACoreDataController*, *TNADataExchange*, *TNAAFClient* and *TNAAppDelegate*. *TNACoreDataController* is used to persistently store the data into the internal storage. Within *TNADataExchange* where all

data management is realized. The *TNAAFClient* is responsible for the communication with the server is the . The *TNAAppDelegate* defines methods that are called by the singleton *UIApplication* object in response to important events in the lifetime of the app [28].

5.1.2 Table View Cells

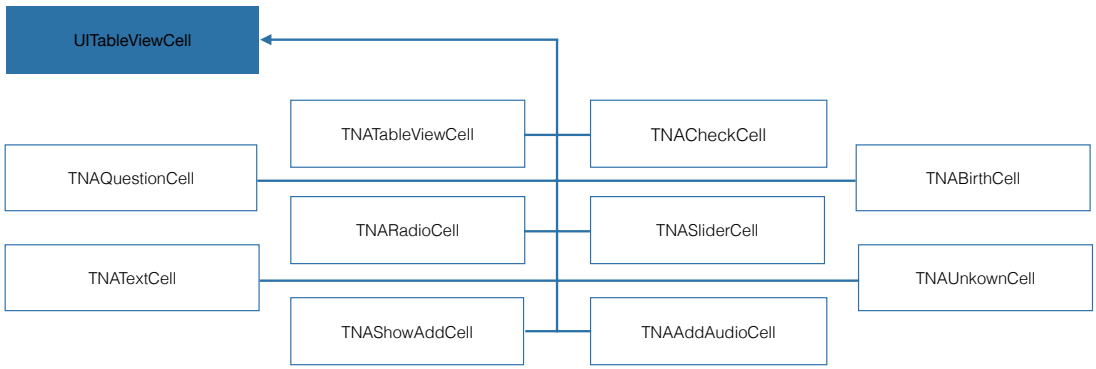

Figure 5.2: Table view cells diagram

Each row of a table in a table view controller is represented by an object of the class *UITableViewCell*. All used sub classes of *UITableViewCell* are displayed in figure 5.2. As mentioned above, the table views of the tabs for *Events*, *Questionnaires*, *Therapies* and *Audio* are managed together. The contents of those table views are dynamic which means the number of rows depends on the data is to be shown. All rows have two text labels which are defined in the class *TNATableViewCell*. For each answer type a class is defined. The *TNATextCell* shows an input text view for multi line answers. There is a button with a label defined to display previously unspecified radio buttons in *TNARadioCell*. Check button answer types are realized with *TNACheckCell*. Like the *TNARadioCell* the *TNACheckCell* has a label and a button. These two answer types have two different classes for easier handling purposes. *TNASliderCell* pictures an answer type *int scale* which is used for selecting a single value from a continuous range of values. The *TNABirthCell* stands for a single line user input which gets a date picker in the *TNAQuestionsTableViewController*. In case there are new questionnaires with

answer types which are not specified yet, *TNAUnknownCell* is catching this case to prevent the app from crashing.
Presenting details of an audiometric measurement requires three labels for each object to display frequency, value of left ear and value of right ear which is implemented in the *TNAShowAudioCell* whereas *TNAAddAudioCell* has only one label for the frequency and two text input fields since this class is designed for adding a new measurement.

5.1.3 Data Model

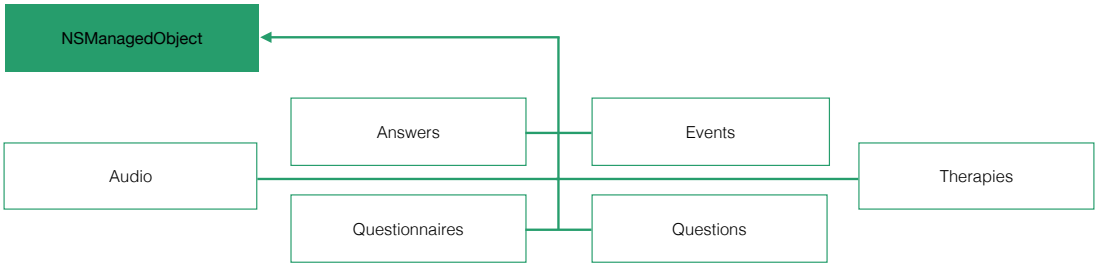

Figure 5.3: Data model diagram

TinnitusNavigator uses the Core Data framework to manage its model objects (cf. 6.3). The data model is based on the JSON (JavaScript Object Notation) objects which are received from the server and are extended with attributes that are used for realizing the data exchange. All classes are subclasses of *NSManagedObject*, which implements all the basic behaviour required of a Core Data model object which the class diagram in figure 5.3 is showing. An object of the class *Events* contains fields for time, title, type, description, updated_at, id, created_at and remote_id and additional objectId as well as exchangeState of one event. For a therapy the attributes name, start_date, end_date, type, note, updated_at, id, created_at and remote_id and additional objectId as well as exchangeState are saved in *Therapies*. The *Audio* class contains time, value_left, value_right, frequency updated_at, id, created_at, remote_id and additional objectId as well as exchangeState properties for each result of a measurement. *Questionnaires* contains following attributes: configuration, which describes the answer possibilities, if all of them are the same as the questionnaire, title, id, created_at, remote_id and

additionally objectId as well as exchangeState. All questions of all questionnaires are stored in *Questions*. They have a configuration (similar to answer possibilities), if they are not already configured in *Questionnaires*, which includes id, position for the order to display, question, questionnaire_id for mapping, type of the answer and additional objectId as well as exchangeState.

5.2 Relation between app and server

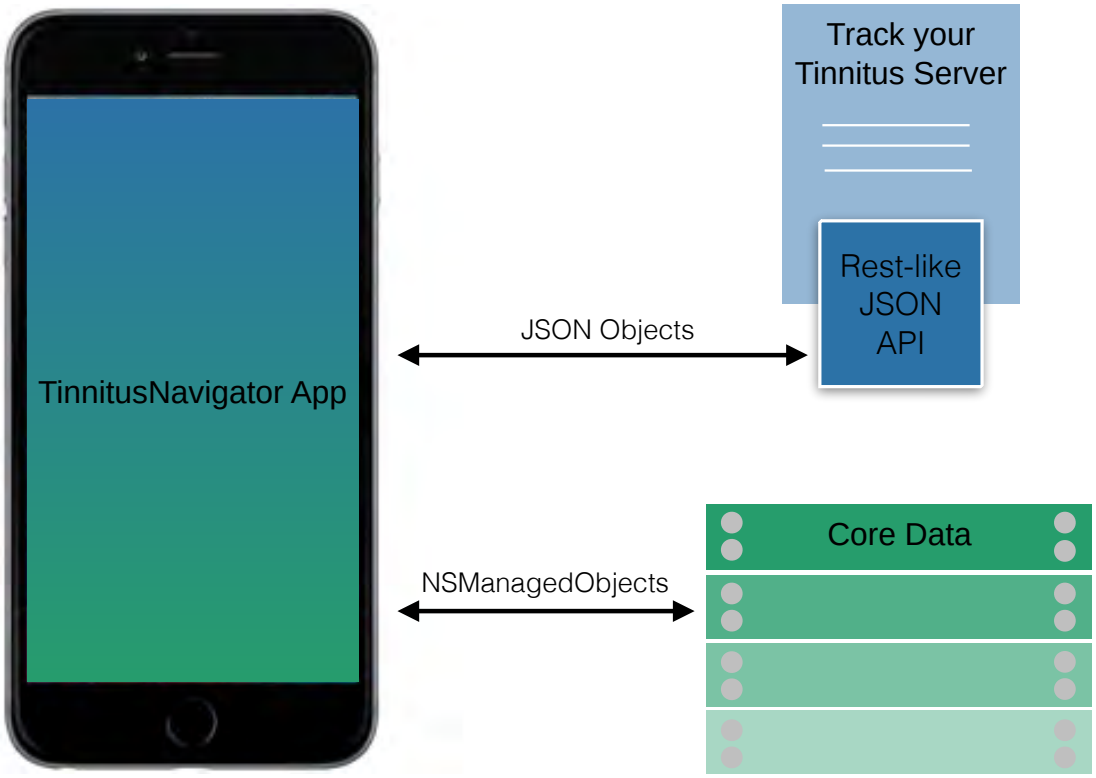

Figure 5.4: Relation between app and server [29]

The TinnitusNavigator communicates with the TrackYourTinnitus server. On the server side all questionnaires and optionally data of events, therapies or audiometric measurements. The user has the option to use the website or to use an app. All communications

between the app and the server is done over a *REST-like JSON API*. A *JSON* is a data-interchange format. The advantage is that it is simple for humans to read and write but also simple for machines to parse and generate. *JSON* provides objects or array of objects which need to be parsed to objective-c objects in order to be processed and saved into Core Data. This action is called *serialization* or *deserialization* if it is the other way around. A detailed description of how this is done in the app is presented in 6 and the figure 5.4 illustrates the described process.

6

Implementation and implementation aspects

This chapter presents how the exchange process of the app and the server takes place. Afterwards the design criteria are described as well as the used frameworks. Finally, the challenges and findings of the implementation are summarized.

6.1 Exchange process with data persisting

An app that works only when connected to the internet is not very practical. Consequently this aspect was included from the beginning of the implementation.

The app uses the *AFNetworking* framework for the communication with the *API*, which makes tasks like asynchronous *HTTP* requests easier to handle. **TNAAFClient** is a

subclass of **AFHTTPRequestOperationManager** which origins from the framework and is using the singleton pattern. A singleton pattern guarantees that only one instance is alive for a given class and that there is a global access point to that instance [30].

Managing all of the exchange routines, between Core Data and the server is the job of the**TNADataExchange** class which is also a singleton class and **NSManagedObject** sub-classes are registered with it. The routine is the process to take data from the server, parse it and save it into Core Data. All registered classes are stored in an array.

In consideration of the user's data plan, the goal is to make use of every call in the most efficient way possible and make sure that every piece of data pulled over the mobile networks counts. Therefore it does not make sense to download and deal with every record each time the exchange process is performed. It makes more sense to compare the *updatedAt* attribute on the entities and determine which is the most recent one. Then to ask the server by using this date to only send those which were modified after this date.

It is important to keep track of all exchange actions as each should not start more than once. The method *startExchange* (c.f. listing 6.1) first checks if an exchange is already running. If not, it sets the property *exchangeInProgress* to *YES*.

```
1  - (void)startExchange {
2    if (!self.exchangeInProgress) {
3      [self willChangeValueForKey:@"exchangeInProgress"];
4      _exchangeInProgress = YES;
5      [self didChangeValueForKey:@"exchangeInProgress"];
6      self.backgroundExchangeQueue = dispatch_get_global_queue(
       DISPATCH_QUEUE_PRIORITY_BACKGROUND, 0);
7      dispatch_async(self.backgroundExchangeQueue, ^{
8      [self requestDataOfClasses:YES toRemoveOffline:NO];
9      });
10   }
11 }
```

Listing 6.1: Method *startExchange*

With the use of *Grand Central Dispatch* (GCD), which runs the processing tasks in the background [31], it kicks off an asynchronous block that calls the method *requestDataOf-*

Classes: toRemoveOffline:. Objects which are deleted on the server and exist locally should also be deleted. Therefore the method *processJSONDataRecordsForRemoval* compares the *JSON* response with all locally stored records and those which can not be matched are deleted. To decrease network traffic even more, as many operations as possible are performed in a single batch with a queue of all requests. The methods *applicationCacheDirectory* and *JSONDataRecordsDirectory* return a location where to save the files and the method *writeJSONResponse* (c.f. listing 6.2) stores the responses to that location before processing them.

```
1  - (void)writeJSONResponse:(id)response toDiskForClassWithName:(NSString *)
       className{
2  NSURL *fileURL = [NSURL URLWithString:className relativeToURL:[self
       JSONDataRecordsDirectory]];
3  if (![(NSDictionary *)response writeToFile:[fileURL path] atomically:YES]){
4    NSArray *records = [response objectForKey:@"results"];
5    NSMutableArray *nullFreeRecords = [NSMutableArray array];
6    for (NSDictionary *record in records){
7       NSMutableDictionary *nullFreeRecord = [NSMutableDictionary
       dictionaryWithDictionary:record];
8       [record enumerateKeysAndObjectsUsingBlock:^(id key, id obj, BOOL *stop){
9          if ([obj isKindOfClass:[NSNull class]]){
10            [nullFreeRecord setValue:nil forKey:key];
11         }
12      }];
13      [nullFreeRecords addObject:nullFreeRecord];
14    }
15    NSDictionary *nullFreeDictionary = [NSDictionary dictionaryWithObject:
       nullFreeRecords forKey:@"results"];
16    if (![nullFreeDictionary writeToFile:[fileURL path] atomically:YES]) {
17      NSLog(@"Failed all attempts to save response to disk: %@", response);
18    }
19   }
20 }
```

Listing 6.2: Method *writeJSONResponse:toDiskForClassWithName:*

In order to know if the exchange is finished and if it is the first time, there are two constants *kTNDateExchangeInitialCompleteKey* and *kTNDataExchangeCompletedNoti-*

ficationName as well as two methods *initialExchangeComplete* and *setInitialExchangeCompleted* which track this information. Furthermore the property *exchangeInProgress* is set to *NO* in the method *executeExchangeCompletedOperations*.

Until that moment the data is stored in a property list format rather than into Core Data. Method *JSONDictionaryForClassWithName* gets the files from disk and returns an NSDictionary, but NSArrays are easier for handling purposes. Consequently the method *JSONDictionaryForClass* calls the previous method and returns an NSArray of all records in the response that are ordered by a given key.

The *JSON* responses are no longer required and the method *removeJSONDataRecordsForClassWithName* deletes them.

During conversion of *JSON* values to objective-c properties the dates are handled in a special way using the *NSDateFormatter* is, which is costly and is therefore re-usable. Following three methods implement the converting:

- *initializeDateFormatter*: is for initialization
- *dateUsingStringFromAPI*: gets an NSString object and returns an NSDate object
- *dateStringForAPIUsingDate*: gets an NSDate object and returns an NSString object

These methods are used in *setValue:forKey:forManagedObject:* which gets a value, key and managedObject and converts them if the key is equal to *created_at*, *updated_at*, *time*, *start_date*, *end_date*, *_time* or *description*. Otherwise the values stay as they are, set in the managedObject.

The enum *TNAObjectExchangeState* has the variables *TNAObjectExchanged*, *TNAObjectCreated* and *TNAObjectDeleted*. The flag *TNAObjectCreated* in the enum *TNAObjectExchangeState* indicates which objects are created locally and need to be pushed to the server. *TNAObjectDeleted* flag is needed to mark local records for deletion.

A new NSManagedObject is created in the *backgroundManagedObjectContext* with the method *createManagedObjectWithClassName:forRecord* (c.f. listing 6.3), which gets a class name and a record from the *JSON* response.

```
1 - (void)createManagedObjectWithClassName:(NSString *)className forRecord:(
    NSDictionary *)record {
2 NSManagedObject *newManagedObject = [NSEntityDescription
    insertNewObjectForEntityForName:className inManagedObjectContext:[[
    TNCoreDataController sharedInstance] backgroundManagedObjectContext]];
3 [record enumerateKeysAndObjectsUsingBlock:^(id key, id obj, BOOL *stop) {
4   [self setValue:obj forKey:key forManagedObject:newManagedObject];
5 }];
6 [record setValue:[NSNumber numberWithInt:TNObjectExchanged] forKey:@"
    exchangeState"];
7 }
```

Listing 6.3: Method *createManagedObjectWithClassName*

Sorting managedObject classes for a defined exchange state is needed to post the right data to server or to delete data on server. For this purpose, the method *managedObjectForClass:withExchangeState* is implemented and called from the methods *postOfflineObjectsToServer* and *removeObjectsFromServer*. The method *managedObjectsForClass:sortedByKey:usingArrayOfIds* works in similar way, excepts it is for processing JSON data records. It gets a class name, a defined key for sorting and an array of objectIds and returns an NSArray of NSManagedObjects. Used is this method of *processJSONDataRecordsIntoCoreData* and *processJSONDataRecordsForRemoval*.

After all *HTTP* requests are finished, the response is written into Core Data and can be grabbed on demand. The registration for the exchange complete notifications is stored *TNATableViewController*. After the view is loaded, method *viewDidAppear* is called and adds an observer for the notification. The observer is again removed in method *viewDidDisappear*.

The classes **TNAAddEventsViewController**,**TNAAddTherapiesViewController** and **TNAAddAudioViewController** each have the method *saveButtonTouched* which sets the *exchangeState* flag in the enum if a new record is added.

Before the data can be sent to the server, they have to be translated into JSON objects with the class *NSManagedObject+JSON* because all classes have different JSON objects. The class contains the methods *JSONToCreateObjectOnServer* for generating

the JSON object and *dateStringForAPIUsingDate* for converting the NSDate into a NSString. *JSONToCreateObjectOnServer* returns an NSDictionary which contains the JSON value required to create the object on the remote service.

AFNetworking converts the NSDictionary to a string for the *POST* request to the server. The *postOfflineObjectsToServer* method initiates the post procedure. If a record is deleted locally the *TNAObjectDeleted* flag is set and the method *removeObjectsOnServer* fetches all objects with this flag and executes the deletion.

There is a refresh button in each view of the **TNATableViewController** class is a refresh button is placed, which allows the user to manually update the data. Observers for changes in class **TNADataExchange** are registered with the implementation of *observeValueForKeyPath:ofObject:change:context:*. In case changes are made, the observer calls the method *checkExchangeState* that questions the **TNADataExchange**-singleton to see if an exchange is in progress. If this is true, method *replaceRefreshButtonWithActivityIndicator* is called and the refresh button shows an indicator. Otherwise the indicator is removed and the refresh icon is shown.

6.2 Design

The iOS design guidelines describe what has to be considered while designing this app [32]. For an easy and smooth navigation in the app, the chosen structure for the TinnitusNavigator app is flat. This app has a lot of information to display which have to be clearly arranged. Due to the tab view all menu items are accessible from the main screen and the user does not have to navigate through a lot of screens to reach the desired destination. Also the user can not get lost as the user's path is logical, predictable and easy to follow. This navigation style is also used in the Apple App Store.

The key colour is the blue tone of the TrackYourTinnitus app which is used in the TinnitusNavigator app to indicate interactive elements and also to show the closeness of both.
Although data from the server is not always well structured (for example, for the result of an audiometry measurement), the user interfaces always keeps the same clarity and a

uniform appearance throughout the app.
Radio buttons and check boxes are not standard in iOS and have to be implemented explicitly. The characteristic of a radio button is that is allows a single selection while a check box allows multiple selections. Both are implemented as normal buttons which changes their image to indicate the selection.

6.3 Used frameworks

Cocoa Pods is the dependency manager used to include third party libraries in this iOS application [33].
Without Cocoa Pods, it would be harder to achieve the same results because the following steps would need to be separately coded: download and unzip files, drag frameworks and bundles to the project, link with new iOS libraries and add linker flags. If the library has dependence on other libraries all of the steps have to be done again. Also updates have to be done manually which would mean that the above described procedure has to be done once more.
With cocoa pods, the author of a library simple writes a *podspec* file that specifies what the developer needs for the proper installation, such as which files are required, where the source location is, what iOS libraries need to run it, pre or post processing stops and so on. The key information required is only its name and perhaps a specific version if needed.

```
1 platform :ios, '7.0'
2 pod 'AFNetworking', '~> 2.2'
3 pod 'AFOAuth2Manager'
```

Listing 6.4: Podfile of the TinnitusNavigator app

The *podfile*, which is written by the developer of the app, lists the names of the podspecs that represents the libraries which will be used in the application. Cocoa Pods takes care of the rest.
There are over 8000 podspec files in the official Cocoa Pods repository which gets automatically updated and all included ones can be easily updated together. To include

Cocoa Pods into Xcode a project has to be created and closed afterwards. In the same directory as the Xcode project the podfile needs to be created.
First, the platform and its minimum deployment target, because libraries might only work on certain versions of iOS, have to be entered. Next the *podspecs* which are needed can be listed (c.f. listing 6.4), (for example, for his app *AFNetworking* and *AFOAuth2Manager*which are described in details in the next section would be listed) .
Cocoa Pods then grabs and installs the libraries which are needed. It creates a xcworkspace file that is a collection of projects and is used from then on for further implementations. There is the original application, that is an empty application which was created earlier and the pods projects which builds all of the Cocoa Pods as static libraries. Cocoa Pods alters the original project to rely on the static libraries built by this other project.

AFNetworking is used for network operations and helps with communication management, serialization, reachability, security and UIKit integration [34].

```
1  - (AFHTTPRequestOperation *)GETAPIRequestForClass:(NSString *)className
2  parameters:(NSDictionary *)parameters
3  success:(SuccessBlockType)success
4  failure:(FailureBlockType)failure {
5  AFOAuthCredential *credential = [AFOAuthCredential
       retrieveCredentialWithIdentifier:@"TYTCredentials"];
6  NSString *access_token = credential.accessToken;
7  NSString *classNameLow = [className lowercaseString];
8  NSString *apiClass = @"api/";
9  NSString *class = [apiClass stringByAppendingString:classNameLow];
10 AFHTTPRequestOperation *operation = [self GET:[NSString stringWithFormat:@"%@
       ?access_token=%@", class, access_token] parameters:parameters success:
       success failure:failure];
11 return operation;
12 }
```

Listing 6.5: GET request with the use of AFNetworking framework

The *AFHTTPRequestOperationManager* class is used in this app in the *TNAAFClient* class for creating the http requests like *GET* (c.f. listing 6.5) and *POST* (c.f. listing 6.6)

as well as serializing the responses.

The serialization module of *AFNetworking* makes it very easy to serialize a request before executing it, by encoding the parameters that need to be included in that request and also helps decoding the response to a specified type. The **AFHTTPRequestOperation** class in the classes**TNADataExchange** and in**TNAWelcomeViewController** it creates and manages the NSURLSession based on a specific configuration and passes to class **TNAAFClient**.

```
1  - (AFHTTPRequestOperation *)POSTAPIRequestForClass:(NSString *)className
2  parameters:(NSDictionary *)parameters
3  success:(SuccessBlockType)success
4  failure:(FailureBlockType)failure {
5  AFHTTPRequestOperation *operation = nil;
6  NSString *classNameLow = [className lowercaseString];
7  AFOAuthCredential *credential = [AFOAuthCredential
       retrieveCredentialWithIdentifier:@"TYTCredentials"];
8  NSString *access_token = credential.accessToken;
9  operation = [self POST:[NSString stringWithFormat:@"api/%@?access_token=%@",
       classNameLow, access_token]
10 parameters:parameters
11 success:success
12 failure:failure];
13 return operation;
14 }
```

Listing 6.6: POST request with the use of AFNetworking framework

AFOAuth2Manager simplifies the process of authenticating against an *OAuth 2.0* provider for third party applications [35]. *OAuth 2.0* is a fundamental component which puts the user in control of its data on the server [36]. If the user sends the user name and password with the traditional model with each request, the credentials have to be validated each time and also be saved in the application which is a security weakness.

With *OAuth 2.0* the user gets an access token from the server, which is a string and has a specific scope, expiration time and other access attributes. There are four roles defined for the *OAuth authorization* and in the following applied to the TinnitusNavigator app:

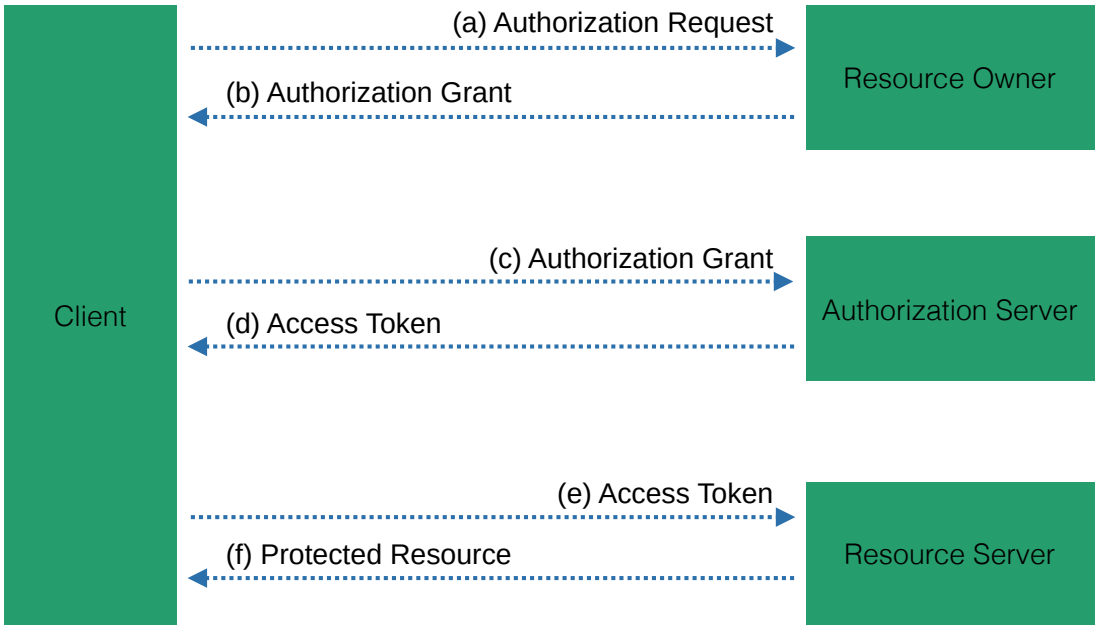

Figure 6.1: Abstract authorization flow [36]

1. **Resource owner:**
   The user of the TinnitusNavigator app, who is able to grant access to a resource.

2. **Resource server:**
   The TrackYourTinnitus server that is hosting the resources, capable of accepting and responding to protected resource requests using access tokens.

3. **Client:**
   The TinnitusNavigator makes protected resource requests in instruction of the user and with its authorization.

4. **Authorization server:**
   The TrackYourTinnitus server is also issuing access tokens to the TinnitusNavigator app after successfully authenticating the user and obtaining authorization.

As can be seen above the authorization server is the same server as the resource server in the TinnitusNavigator application.

The image 6.1 illustrates the interaction between the app and the server and includes the following steps:

(a) The app requests authorization from the user. The authorization request is the login screen of the app.

(b) The app receives an authorization grant, which is a credential representing the user's authorization, expressed using the grant type user password credentials.

(c) The app requests an access token by authenticating with the server and presenting the authorization grant.

(d) The server authenticates the app by the provided client_id with a client_password and validates the authorization grant, and if valid, issues an access token.

No user's credentials are persisted on the device. Only the access token is saved and needed for the data exchange with the server.

(e) The app requests the protected resource from the server and authenticates by presenting the access token.

(f) The server validates the access token, and if valid, serves the request. Otherwise, the access token became invalid and the user needs to authenticate again.

The implementation of the authorization is displayed in listing 6.7.

```
- (IBAction)Login{
...
//GET ACCESS_TOKEN
NSURL *baseURL = [NSURL URLWithString:@"https://secure.dbis.info/ma/tinnitusk
    /login"];
AFOAuth2Manager *OAuth2Manager = [[AFOAuth2Manager alloc] initWithBaseURL:
    baseURL clientID:@"124324324" secret:@"123123"];

[OAuth2Manager authenticateUsingOAuthWithURLString:@"https://secure.dbis.info
    /ma/tinnitusk/oauth/access_token/"
      username:username password:password scope:@"" success:^(
    AFOAuthCredential *credential) {
        self.access_token = credential.accessToken;
        ...
        [AFOAuthCredential storeCredential:credential withIdentifier:@"
    TYTCredentials"];
        ...
        }
      failure:^(NSError *error) {
        self.infoMessage.text = NSLocalizedString(@"Try again", nil);
        ...
        }];
username = nil;
password = nil;
self.infoMessage.text = infoMessage;
}
```

Listing 6.7: Requesting the access token procedure

Core Data is a framework from Apple which is integrated with the *Core Foundation* framework [37]. Therefore no i is needed for this framework. It works as a database although it is not a database. Core Data is the model layer of the TinnitusNavigatorn app in the Model-View-Controller pattern and it manages an object graph. That object graph is persisted by writing it to disk. The heart of the framework is the Core Data stack which is a collection of objects. Managed object model, persistent store coordinator and

managed object contexts are the key objects of the stack. The stack is illustrated in figure 6.2

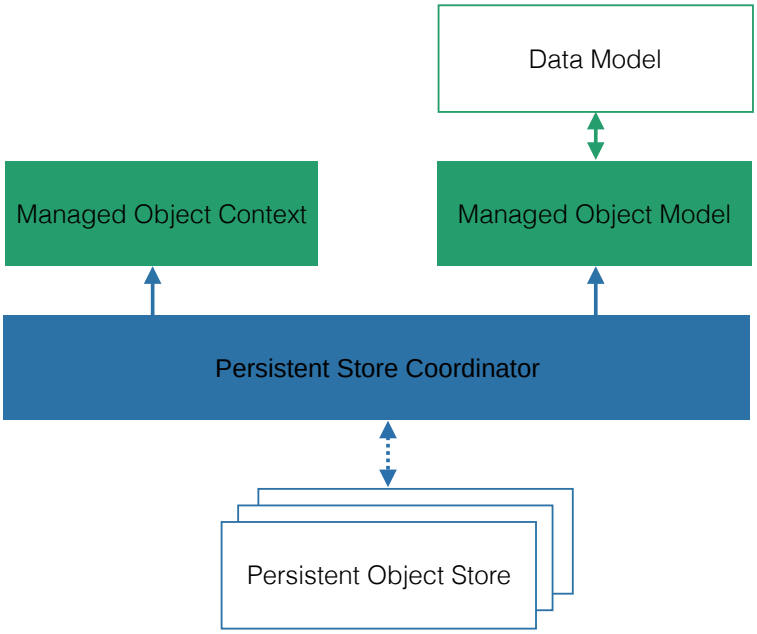

Figure 6.2: Core Data Stack [37]

The managed object model is the data model and includes information about the models or entities of the object graph, attributes and the relation to each other. The **NSManagedObjectModel** object knows about the data model by loading one or more data model files during its initialization.
With the **NSPersistentStoreCoordinator** object data is saved, loaded and cached to disk and guarantees the persistent store(s) and the data model match. It communicates between the persistent store(s) and the managed object context(s). **NSManagedObjectContext** object administers a range of model objects, entities of the **NSManagedObject** class. Every managed object context is secured by a persistent store coordinator. Therefore multiple managed object contexts are feasible.

Data Model contains entities, attributes and relationships. Entities can be compared to a table in a database and attributes are like a column of a table. Relationships are only loaded if they are needed in the application. A record is represented in Core Data's

backing store by instances of **NSManagedObject** like a row in a database table. Each **NSManagedObject** instance is linked with an instance of **NSEntityDescription**. The entity description includes information about the managed object, such as the entity of the managed object, its attributes and relationships. A managed object is also associated to an instance of **NSManagedObjectContext**. The managed object context to which a managed object belongs to, monitors the managed object for changes.

6.4 Challenges and findings

During the implementation a couple of problems occurred. The app uses the same server as the Android app, which is still under construction as mentioned in section 2.5 and was the major challenge during the implementation. In the API not all needed functions were defined at the beginning like the one for the therapies or for deleting any records. Additionally, some received data types are not consistent, for example the date which is in certain cases a normal date or the date in milliseconds. That was a pitfall while implementing the events or therapies part and had to be considered as well handled.

On the server side the function for the registration is faulty. After all data for the registration is typed in complete, the server should send a activation link to the new user, but an old and inactive email address is deposited. As a result, no new user can create an account also can not reset a password due to the same fault. This is only possible, if someone with the rights on the server activates the account or resets the password manually

Towards the end of the implementation, to the data model on the server a further attribute called "remote_id" was added which identifies an object in the server database and makes is easier to modify or delete objects on server side. This was not a big problem as the app used an attribute "object_id" with handled the local data management and only had to be adjusted.

Another challenge was the way the data is stored in the objects. Events or therapies are saved in an object each. On the other hand the data of an audiometry measurement are

distributed in several objects with each containing a time, frequency, value of the left ear and value of the right ear. That situation was solved fast, because of the use of Core Data which made fetching the wanted data simple.

On the subject of Core Data, it was one of the findings during the implementation. It supported to keep everything modular for extension and simplified the communication with server. Also it saved a lot of time with the data management on the app and requesting the data from the server.

A great finding as well was the tables with dynamic prototype with custom cells that are likewise helping the modular feature of the app. Especially for the realisation of the questionnaires they were a big advantage. With them, the tables can be configured as needed, the cells can be reused and keep the designing light.

7

Requirements Comparison

This chapter describes the requirements that have been defined in chapter 3, compared to the implementation and functions of the app. Like in the definition of the requirements this chapter is divided into functional and non-functional requirements as well as a summary in table 7.1.

7.1 Functional requirements

This section shows the comparison of the functional requirements of the app. It checks whether all the functional requirements have been met in the current implementation.

FR 1: **Sign up** - fulfilled

The user is able to register inside the app as the mobile sign up website is opened in a web view, but an error on the server-side prevents to complete the process.

FR 2: **Log in** - fulfilled

By providing a user name and a password the user can log in through the app. For the authorization is OAuth 2.0 used.

FR 3: **Log out** - fulfilled

In the More tab the user has the possibility to logout. With the logout the access token is deleted and the log in view is shown again.

FR 4: **Reset password** - fulfilled

Like for the sign up there is a web view with the mobile reset password website, due to the mentioned server-side problem, this function is not working as well.

FR 5: **Operate without internet connection** - fulfilled

The app checks for the internet connection before every request. If there is none, the user sees an error message about this and can continue working in the app.

FR 6: **Synchronize local data with server data** - fulfilled

Local created data and data which could not be send to the server due to missing internet connection get flagged to be exchanged when the internet connection is established (again).

FR 7: **Display all kinds of questionnaires from server in app** - fulfilled

On basis of the answer types from each question every questionnaire can be built. If there will be a new answer type in the future, the app does not crash instead it shows that these answer type is not supported yet. That prevents the app from being unusable.

FR 8: **Fill out questionnaires** - partially fulfilled

The user can fill out questionnaires and if all questions are answered completely, they can be saved at the end. Yet, the indicator that signals which questionnaires are already filled out is missing.

FR 9: **Enter/show/delete events** - partially fulfilled

In the events tab, the user can add a new one with providing a title, date from the date picker, type of the item picker and an optional description. To delete an event, the user has to swipe the one in the overview to the left. For the details view of

an event, the event has to be selected as well from the overview. Displaying a calender with all entered events is not implemented.

FR 10: **Enter/show/delete therapies** - fulfilled
Entering, showing and deleting is the same procedure as for the events. Therapies consist of a name, start date & end date from a date picker, a type of a item picker and an optional note.

FR 11: **Enter/show/delete measurements of audiometry** - partially fulfilled
It is the same for entering, showing and deleting measurements of audiometries. A measurement is made of a date and values of both ears for eight defined frequencies. What is missing are a diagram for each ear and that the user can add additional frequencies.

FR 12: **Update data automatically and manually** - fulfilled
It is monitored if data is changing and in that case an exchange with the server is activated. Further the user has the option to push a refresh button.

7.2 Non-functional requirements

Next, the requirements comparison with the non-functional requirements are following.

NFR 1: **No saving of email or ip addresses** - fulfilled
The app only saves an access token after a successful log in. Entered user names, email addresses or password are only needed for obtaining the access token and never persisted.

NFR 2: **Use colour of TrackYourTinnitus app** - fulfilled
The key colour for user interaction is the same blue tone as from the TrackYourTinnitus app. Also the background colours are the TrackYourTinnitus' blue and green.

NFR 3: **Following iOS design guidelines** - fulfilled
Like in the iOS design guidelines the TinnitusNavigator app has a flat navigation, the menu items are arrange in a tab view for a fast access and one key colour is used.

NFR 4: **Intuitive user interface** - fulfilled

Not only the tab view provides a intuitive and time saving navigation but it is also known for use from the Apple App Store.

NFR 5: **Make app scalable as much as possible** - fulfilled

It does not matter how many events, therapies, questionnaires or audiometry measurements a user has, the app is designed with the table view to list the data in them.

NFR 6: **Release in Apple App Store** - not fulfilled

This requirement could not be achieved due to incomplete functionality of the TinnitusNavigator app and missing methods on the server.

No.	Description	Fulfilled?
FR 1	Sign up	yes
FR 2	Log in	yes
FR 3	Log out	yes
FR 4	Reset password	yes
FR 5	Operate without internet connection	yes
FR 6	Synchronize local data with server data	yes
FR 7	Display all kinds of questionnaires from server in app	yes
FR 8	Fill out questionnaires	partially
FR 9	Enter/show/delete events	partially
FR 10	Enter/show/delete therapies	yes
FR 11	Enter/show/delete measurements of audiometry	partially
FR 12	Update data automatically and manually	yes
NFR 1	No saving of email addresses or ip addresses	yes
NFR 2	Use colour of Track Your Tinnitus project	yes
NFR 3	Following iOS design guidelines	yes
NFR 4	Intuitive user interface	yes
NFR 5	Make app scalable as much as possible	yes
NFR 6	Release in Apple App Store	no

Table 7.1: Summarizing table with all comparisons

8

Outlook

With termination of this thesis, the development of the TinnitusNavigator app is not yet completed. There are some ideas for the future in addition to the requirements which can be developed afterwards.

For example, it would be comfortable to be able to search on the tables for certain data for a faster access. Therefore, a search bar could be added above each table where the keyword can be typed in and the table only displays the corresponding entries.

Currently registered data can not be changed after they are saved. Thus, the data do not always need to be deleted and recreated. It would be better to simply change those entries. The above mentioned search function would be very useful in this case.

Another idea is a user specific sort of therapies and events, that way the app is even more personalized and the focus stays on what is important to the user. Sort options could be the name, date or time of the entries.

The TinnitusNavigator app could also remember the user that the end date of a therapy, if set, is coming closer so the necessary action can be taken. A reminder can also be used, if the app was not used for a long period, to insert new data and profit from it.

Before those ideas are implemented, the not entirely fulfilled requirements need to be completed. The calendar view for the events has not yet been implemented. Also the entered auditory measurements should be visualized in diagrams. For adding a new result of audiometry, the user needs to be able to add values for custom frequencies. After all, adding the indicators of the questionnaires which are already completed, will finalize the app. To ensure the TinnitusNavigator app is working on all target devices, it should be tested, as it was only checked with the simulator and on an iPhone6 Plus.

Is the TinnitusNavigator app completed and everything is fixed on the server side, like the register and reset password functions, it is time to release it in the Apple App store to be available for patients.

9

Conclusion

This master's thesis deals with the concept and implementation of a mobile health record application for tinnitus patients. The focus was to develop a scalable app which can keep events, therapies, questionnaires and audiometry measurements online as well as offline. It is successful implemented with the TinnitusNavigator app.

During forming the user interface a lot of attention was spent on the operation concepts of the iOS User Guideline to provide a short familiarization period. Due to the tab menu, the TinnitusNavigator app has a flat navigation. It enables a fast selecting of menu items and the entries can be deleted with a swipe gesture. New events, therapies and audiometry measurements can easily be added and are shown in an overview table with their representing key words. Also questionnaires loaded from the server can be comfortable filled out inside the app and the answered ones are saved. Despite of the incomplete server at the beginning of this thesis, the functionality of the TinnitusNavigator

app is achieved anyway. The dominate colours are blue and green tones, that are the colours like in the TrackYourTinnitus app to symbolize the affiliation. A new experience was to work with the Core Data framework that was a great help to realize the model view controller pattern. It was challenging to figure out how the principle of the exchange between the TinnitusNavigator app and the server should work and how to integrate this in the application. As soon as all issues from the previous chapter are solved, the TinnitusNavigator app will be published in the Apple App Store.

Bibliography

[1] Pryss, R., Langer, D., Reichert, M., Hallerbach, A.: Mobile task management for medical ward rounds - the medo approach. In: 1st Int'l Workshop on Adaptive Case Management (ACM'12), BPM'12 Workshops. Number 132 in LNBIP, Springer (2012) 43–54

[2] Pryss, R., Mundbrod, N., Langer, D., Reichert, M.: Supporting medical ward rounds through mobile task and process management. Information Systems and e-Business Management **13** (2015) 107–146

[3] Schobel, J., Schickler, M., Pryss, R., Nienhaus, H., Reichert, M.: Using vital sensors in mobile healthcare business applications: Challenges, examples, lessons learned. In: 9th Int'l Conference on Web Information Systems and Technologies (WEBIST 2013), Special Session on Business Apps. (2013) 509–518

[4] Jastreboff, P.J.: Phantom auditory perception (tinnitus): mechanisms of generation and perception. Neuroscience research **8** (1990) 221–254

[5] Lehnhardt, E., Laszig, R.: Praxis der Audiometrie. Georg Thieme Verlag (2009)

[6] Baguley, D., McFerran, D., Hall, D.: Tinnitus. The Lancet **382** (2013) 1600–1607

[7] Hesse, G.: Tinnitus. Georg Thieme Verlag (2008)

[8] Goebel, G., Hiller, W.: Qualitätsmanagement in der therapie des chronischen tinnitus. Oto-Rhino-Laryngologia Nova **10** (2000) 260–268

[9] Schobel, J., Schickler, M., Pryss, R., Maier, F., Reichert, M.: Towards process-driven mobile data collection applications: Requirements, challenges, lessons

learned. In: 10th Int'l Conference on Web Information Systems and Technologies (WEBIST 2014), Special Session on Business Apps. (2014)

[10] Crombach, A., Nandi, C., Bambonye, M., Liebrecht, M., Pryss, R., Reichert, M., Elbert, T., Weierstall, R.: Screening for mental disorders in post-conflict regions using computer apps - a feasibility study from burundi. In: XIII Congress of European Society of Traumatic Stress Studies (ESTSS) Conference. (2013) 70–70

[11] Isele, D., Ruf-Leuschner, M., Pryss, R., Schauer, M., Reichert, M., Schobel, J., Schindler, A., Elbert, T.: Detecting adverse childhood experiences with a little help from tablet computers. In: XIII Congress of European Society of Traumatic Stress Studies (ESTSS) Conference. (2013) 69–70

[12] Ruf-Leuschner, M., Pryss, R., Liebrecht, M., Schobel, J., Spyridou, A., Reichert, M., Schauer, M.: Preventing further trauma: Kindex mum screen - assessing and reacting towards psychosocial risk factors in pregnant women with the help of smartphone technologies. In: XIII Congress of European Society of Traumatic Stress Studies (ESTSS) Conference. (2013) 70–70

[13] Schobel, J., Ruf-Leuschner, M., Pryss, R., Reichert, M., Schickler, M., Schauer, M., Weierstall, R., Isele, D., Nandi, C., Elbert, T.: A generic questionnaire framework supporting psychological studies with smartphone technologies. In: XIII Congress of European Society of Traumatic Stress Studies (ESTSS) Conference. (2013) 69–69

[14] Schobel, J., Pryss, R., Reichert, M.: Using smart mobile devices for collecting structured data in clinical trials: Results from a large-scale case study. In: 28th IEEE International Symposium on Computer-Based Medical Systems (CBMS 2015), IEEE Computer Society Press (2015)

[15] Schobel, J., Schickler, M., Pryss, R., Reichert, M.: Process-driven data collection with smart mobile devices. In: Web Information Systems and Technologies - 10th International Conference, WEBIST 2014, Barcelona, Spain, Revised Selected Papers. LNBIP, Springer (2015)

[16] Hiller, W., Goebel, G.: Rapid assessment of tinnitus-related psychological distress using the mini-tq. Int J Audiol **43** (2004) 600–4

[17] Langguth, B., Goodey, R., Azevedo, A., Bjorne, A., Cacace, A., Crocetti, A., Del Bo, L., De Ridder, D., Diges, I., Elbert, T., et al.: Consensus for tinnitus patient assessment and treatment outcome measurement: Tinnitus research initiative meeting, regensburg, july 2006. Progress in brain research **166** (2007) 525–536

[18] Herrmann, J.: Konzeption und technische realisierung eines mobilen frameworks zur unterstützung tinnitusgeschädigter patienten. (2014)

[19] Lindinger, M.: Konzeption und implementierung einer mobilen anwendung zur unterstützung von tinnitus-patienten. (2014)

[20] : Trackyourtinnitus project website. (Online) `https://www.trackyourtinnitus.org/de/home` Last checked: 14.09.15.

[21] Pryss, R., Reichert, M., Herrmann, J., Langguth, B., Schlee, W.: Mobile crowd sensing in clinical and psychological trials ? a case study. In: 28th IEEE Int'l Symposium on Computer-Based Medical Systems. (2015)

[22] Pryss, R., Reichert, M., Langguth, B., Schlee, W.: Mobile crowd sensing services for tinnitus assessment, therapy and research. In: IEEE 4th International Conference on Mobile Services (MS 2015), IEEE Computer Society Press (2015)

[23] Schlee, W., Herrmann, J., Pryss, R., Reichert, M., Langguth, B.: Moment-to-moment variability of the auditory phantom perception in chronic tinnitus. In: 13th Int'l Conf on Cochlear Implants and Other Implantable Auditory Technologies. (2014)

[24] Schlee, W., Herrmann, J., Pryss, R., Reichert, M., Langguth, B.: How dynamic is the continuous tinnitus percept? In: 11th International Tinnitus Seminar. (2014)

[25] : Apple - app store. (Online) `https://developer.apple.com/library/mac/documentation/General/Conceptual/DevPedia-CocoaCore/Singleton.html` Last checked: 14.09.15.

[26] : Apple - defining classes. (Online) `https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/`

ProgrammingWithObjectiveC/DefiningClasses/DefiningClasses.html Last checked: 14.09.15.

[27] : Apple - adding user defined runtime attributes. (Online) https://developer.apple.com/library/mac/recipes/xcode_help-interface_builder/Chapters/AddUserDefinedRuntimeAttributes.html Last checked: 14.09.15.

[28] : Apple - appdelegate reference. (Online) https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIApplicationDelegate_Protocol/index.html Last checked: 14.09.15.

[29] : Apple - iphone6. (Online) http://www.apple.com/shop/buy-iphone/iphone6 Last checked 14.09.15.

[30] : Apple - singleton. (Online) https://developer.apple.com/library/mac/documentation/General/Conceptual/DevPedia-CocoaCore/Singleton.html Last checked: 14.09.15.

[31] : Apple - grand central dispatch (gcd) reference. (Online) https://developer.apple.com/library/mac/documentation/Performance/Reference/GCD_libdispatch_Ref/ Last checked: 14.09.15.

[32] : Apple - ios human interface guidelines. (Online) https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/index.html#//apple_ref/doc/uid/TP40006556-CH66-SW1 Last checked: 14.09.15.

[33] : Cocoapods. (Online) https://cocoapods.org Last checked: 14.09.15.

[34] : Afnetworking. (Online) http://cocoadocs.org/docsets/AFNetworking/2.6.0/ Last checked: 14.09.15.

[35] : Afoauth2manager. (Online) http://cocoadocs.org/docsets/AFOAuth2Manager/2.2.0/ Last checked: 14.09.15.

[36] D. Hardt, E.: Rfc 6749 - the oauth 2.0 authorization framework. Technical report, Internet Engineering Task Force (IETF) (2012)

[37] : Apple - core data programming guide. (Online) https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/CoreData/CoreData.pdf Last checked: 14.09.15.

List of Figures

List of Tables

Listings

Name: Carmen Vazinkhoo Matrikelnummer: 668633

Erklärung

Ich erkläre, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den .....

Carmen Vazinkhoo